BEPI USER MANUAL

General

This manual is intended for those with little or no experience of programming. More experienced programmers may prefer to refer to a shorter document “BEPI for Programmers”.

As with any programming, examples are invaluable and these can be downloaded from the EMACSYS web site at www.emacsys.com.

Introduction

BEPI (Basic EMACSYS Programmers Interface) is a simple programming language which enables the user to write applications which can interact with the physical I/O and other non-physical registers. While it capable of running quite sophisticated applications it kept intentionally simple and therefore it does not have the flexibility or complexity of other high level languages.

Because of its simplicity it is very easy to learn and hence well suited to those who have limited or no previous experience of programming.

BEPI Processing

The BEPI program is compiled and then downloaded into the EMACSYS. The appware will then run the compiled code. When it comes to the end of the code it will then return to its main tasks. Once those are complete it will again run the BEPI code and so on.

The period between each run will depend on how much time the main code needs.

Program Structure

The structure of the code is as follows:

```
Variables
/List of variables
End

Start
/Main program
End

sub Name
/Subroutine
subend

sub Name1
/Subroutine
subend
```

The first section defines the type and name of all the variables which are to be used in the program. Next is the main body of the program. Finally is the subroutines which will be used by the main part of the program.

Most applications will need to use some variables but for some simple applications they may not be necessary. In these cases the variables section can be left out.

All applications will need to use the main program section as this is where the code is run.

Subroutines can be used for frequent tasks in order to reduce the amount of main code. It will not always be necessary to incorporate subroutines.

Note that the variables section, if used, must be the first section and subroutines, again if used, must follow the main body.

Variables

Variables are areas of memory which are used to store data. At start up and after downloading a new file all variable data is set to zero. These variable are only accessed by the BEPI processing and are not used by the main appware. Once a value is assigned to a variable it will retain its value until it is changed by the BEPI processing again.

All variables are unsigned. That is their value cannot be negative.

All variable names are case sensitive.

There are three different types of variables. The only real difference between them is the range of values that they can hold:

u8 is 8 bits wide (1 byte) and can hold values in the range 0 to 256.
u16 is 16 bits wide (2 bytes) and can hold values in the range 0 to 65535.
u32 is 32 bits wide (4 bytes) and can hold values in the range 0 to 4294967295.

It is important to choose the correct variable type for its purpose. At first it may seem that it would be better to use u32 for all the variables but these take twice the available space as the u16 type and four times the space as the u8.

Variable names must start with a letter but may contain numbers as well as letters up to a maximum of 32 characters. No characters other than letters and numbers are allowed.

Example of a variables section:

```
Variables
u8 Variable1
u16 Variable2
u32 Variable3
End
```

Note that the section starts with the word “Variables” on a line on its own and ends with “End” again on its own line. Blank lines are permitted within the section.

While this is acceptable as far as the compiler is concerned the names do not convey any meaning. Variable1 only tells us that it is a variable which presumably we know already. It would make for more readable code if we gave the variables names which reflect their use.

```
Variables
u8 Test
u16 InputStatus
u32 Counter
End
```

Variable Arrays

Arrays are much the same as variables but will contain a number of elements. For example:

```
Variables
u8 Test[8]
u16 InputStatus
u32 Counter
End
```

In this case Test will have 8 elements each of type u8. This type of variable can be useful when you need to

check a number of variables which have the same sort of function.

The first element of the array will be indexed as zero (Test[0]) the second as 1 (Test[1]) etc.

Interface Variables

This type of variable is pre-defined and should not be included in the Variable section. Indeed if they are defined in the Variable section the compiler will generate an error.

These variables differ from the standard type in that they may be accessed and used by the main appware. For example “Inputs” will contain the current state of all the physical inputs. By reading this variable and after suitable processing writing to the variable “Outputs” it is possible to control the physical outputs.

As well as reading and writing to all the I/O in a block it is also possible read from or write to single elements of the I/O. For example “Input1” will defines the state of input 1 while “Output2” defines the state of output 2.

As well as well as the physical I/O there is also a virtual variable. In a similar manner to the physical I/O this variable can be accessed as either a full variable or as individual elements. This can be used as a method for a user to alter the behaviour of the BEPI processing via the main appware. For example in the Ethernet Server it is possible to set a value in the virtual register. The BEPI processing can then inspect that value and act accordingly.

Code Elements

Assignments

The simplest form of assignment is to give a variable a fixed value:

Variable1=5

Thus Variable1 will have a value of 5. This can then be passed to another variable;

Variable2=Variable1

In this example after this line of code has been processed the value of Variable2 will be changed to the value of Variable2. Putting the two line together:

Variable1=5
Variable2=Variable1

Both Variable1 and Variable2 will have the value of 5.

If we change this code now to:

Variable1=5
Variable2=Variable1+2

Variable1 will still have the value of 5 but Variable2 will have a value of 7.

We can extend this further:

Variable1=5
Variable2=Variable1+2
Variable3=Variable1+Variable2

In this case Variable3 will have a value of 12 (Variable1=5,Variable2=7).

The second line can also be written as:

Variable2=2+Variable1

Which will give the same result.

Subtraction, multiplication and division are also possible:

```
Variable1=5
Variable2=Variable1-2
Variable3=Variable1-Variable2

Variable1=5
Variable2=Variable1*2
Variable3=Variable1*Variable2

Variable1=5
Variable2=Variable1/2
Variable3=Variable1/Variable2
```

Several arithmetic operations are allowed for each line. Therefore the following line is legal:

```
Variable3=Variable1+Variable2+2
```

However care should be taken where mix arithmetic operations are performed. The BEPI processing will perform each operation in sequence. For example:

```
Variable1=1+2*5
```

will give a result for Variable1 of 15 whereas

```
Variable1=2*5+1
```

will give a result of 11.

Conditional Statements

Conditional statements allow the processing of a section of code if the associated condition is true and optionally a different section of code if it is false.

The simplest form is:

```
if Variable1=1
        Variable2=2
endif
```

In this case Variable2 will be set to 2 if and only if Variable1=1. Note that the value of Variable1 is not changed. The second line is indented. This is not necessary as far as the compiler is concerned but it does make the code easier to read.

Several lines can be run between the if statement and the endif statement:

```
if Variable1=1
        Variable2=2
        Variable3=Variable2-Variable1
        Variable1=5
endif
```

These statements can be nested. That is further if statements can be included in conditional code:

```
if Variable1=1
        if Variable2=2
                Variable3=Variable2-Variable1
                Variable1=5
        endif
endif
```

Three types of test may be applied: '=' as shown above, '>' greater than or '<' less than;

```
if Variable1>1
	Variable2=2
endif
```

In this case Variable2=2 only if Variable1 is greater than 1.

```
if Variable1<1
	Variable2=2
endif
```

In this case Variable2=2 only if Variable1 is less than 1.

These three elements may be combined. >= or => will be true if the variable is greater than or equal to the test value. Similarly <= or =< will be true if the variable is less than or equal to the test value. Finally <> or >< will be true if the variable is not equal to the test value.

The test value does not have to be fixed as in the above examples. It is also possible to use other variables.

```
if Variable1=Variable2
	Variable3=1
endif
```

In this case Variable3=1 only if Variable1 has the same value as Variable2.

You may need to run one set of code if the condition is true and another set of code if it is false. This can be done as follows:

```
if Variable1=1
	Variable2=2
else
	Variable2=3
endif
```

In this case Variable2=2 if Variable1 has a value of 1 otherwise Variable2=3.

Loops

Sometimes it is necessary to run a section of code repetitively. This can be done with a for statement:

```
for 5
	Variable1=Variable1+1
endfor
```

This code will loop 5 times thus increasing the value of Variable1 by 5. As with if statements for loops can be nested up to a maximum of 5 loops.

```
for 5
	Variable1=Variable1+1
	for 2
		Variable2=Variable2+1
	endfor
endfor
```

As with the previous example Variable1 will be increased by 5. Variable2 however will be increased by 10. That is because each time the first loop is run the second loop will be run twice.

It is often useful to use arrays within loops. In the following example six elements of an array are set to zero:

```
Count=0
for 5
	Test[Count]=0
	Count=Count+1
```

endfor

Termination

Generally the BEPI will process all code until it reaches the end of the main section. However there may be cases where it is required to terminate the processing early. In this case a return statement may be used. For example:

```
if Virtual1=0
        return
endif
```

will stop the current processing if Virtual1 has not been set.

Subroutines

Subroutines are sections of code that are used frequently and can thus reduce the total code used. They can only be used from the main code. That is a subroutine cannot call another subroutine.

Each subroutine should have a unique name and are run from the main code with a call statement.

In the main code:

```
call Increment
```

Then the subroutine:

```
sub Increment
Counter=Counter+1
subend
```

All the elements used in the main code can be used in subroutines.

When the BEPI processing reaches the subend line it will return to the line after the original call instruction. In some cases it may be necessary to terminate the subroutine early. This can be done with a return statement. For example:

```
if Variable1>10
        return
endif
```

Subroutines should be placed after the main code.

Comments

It is often useful to put some form of explanation within the code so that when you come back to it after some time it is clear what is happening. This can be done with a '/' as the first character in a line:

```
/This is a comment
```

Commented lines are ignored by the compiler.

APPENDIX 1

INTERFACE VARIABLES

Inputs
Variable type u16.
Value of the physical inputs.
It is possible to write to this register but the new value will only be held until the inputs are read again at which point they will be updated to a value which reflects the value of the physical inputs. If the BEPI processing does change the value it will normally remain unchanged until just before the BEPI processing is applied again.
Read /Write

Input1
Input2
Input3
Input4
Input5
Input6
Input7
Input8
Input9
Input10
Input11
Input12
Input13
Input14
Input15
Input16
Variable type u8.
Value of the individual physical inputs. If the input is active it will read as 1 otherwise it will read as 0.
It is possible to write to these registers but the new value will only be held until the inputs are read again at which point they will be updated to a value which reflects the value of the physical inputs. If the BEPI processing does change the value it will normally remain unchanged until just before the BEPI processing is applied again.
Writing a 1 to the register will set the input to active. Writing a 0 to the register will set the input to inactive.
Read /Write

Outputs
Variable type u16.
Value of the physical outputs.
It is still possible for external parts of the appware to change the value of the output register. However in most applications these will not be sent to the physical outputs until after the BEPI processing.
Read /Write

Output1
Output2
Output3
Output4
Output5
Output6
Output7
Output8
Output9
Output10
Output11
Output12
Output13
Output14
Output15

Output16
Variable type u8.
Value of the individual physical outputs. If the output is active it will read as 1 otherwise it will read as 0.
It is still possible for external parts of the appware to change the value of the output register. However in most applications these will not be sent to the physical outputs until after the BEPI processing.
Writing a 1 to the register will set the output to active. Writing a 0 to the register will set the output to inactive.
Read /Write

Analogue1
Analogue2
Analogue3
Analogue4
Analogue5
Analogue6
Analogue7
Analogue8
Variable type u16.
Value of the physical A/D. The range of these registers will be 0 to 1023.
It is possible to write to these registers but the new value will only be held until the inputs are read again at which point they will be updated to a value which reflects the value of the physical inputs. If the BEPI processing does change the value it will normally remain unchanged until just before the BEPI processing is applied again.
Read /Write

Tick
Variable type u16.
This register is incremented every millisecond and has a maximum value of 65535. Once it has reached this value it will be reset to zero automatically. This register is useful for short timing period.
Note that it is not guaranteed that the BEPI processing will be applied within the 1mS time period. Therefore the value of this register may increase by more than 1 for each BEPI iteration.
Read /Write

Virtuals
Variable type u8.
Value of a non-physical register which is used for interaction between the external appware and the BEPI processing.
Read /Write

Virtual1
Virtual2
Virtual3
Virtual4
Virtual5
Virtual6
Virtual7
Virtual8
Variable type u8.
Value of a non-physical register which is used for interaction between the external appware and the BEPI processing.
Writing a 1 to the register will set the virtual output to active. Writing a 0 to the register will set the virtual output to inactive.
Read /Write

Seconds
Register containing the seconds value of the real time clock.
Read only.

Minutes
Register containing the minutes value of the real time clock.
Read only.

Hours
Register containing the hours value of the real time clock.

Read only.

Days
Register containing the day of the month value of the real time clock.
Read only.

Months
Register containing the month value of the real time clock. 1 = January..... 12 = December.
Read only.

Years
Register containing the year value of the real time clock.
Read only.

Weekday
Register containing the weekday value of the real time clock. 0 = Sunday, 1 = Monday ….. 7 =Saturday.
Read only.

APPENDIX 2

RESERVED WORDS

The following words cannot be used a variables:

return
if
else
endif
for
endfor
call
sub
subend